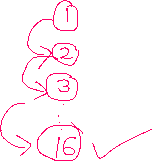
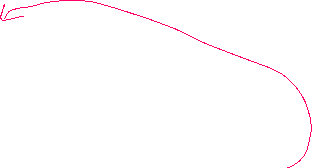
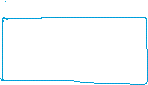
43. Recursive Trigger and Record Sharing- 02 June 2022

1] Recursive Trigger

2] Record Sharing

1] Recursive Trigger

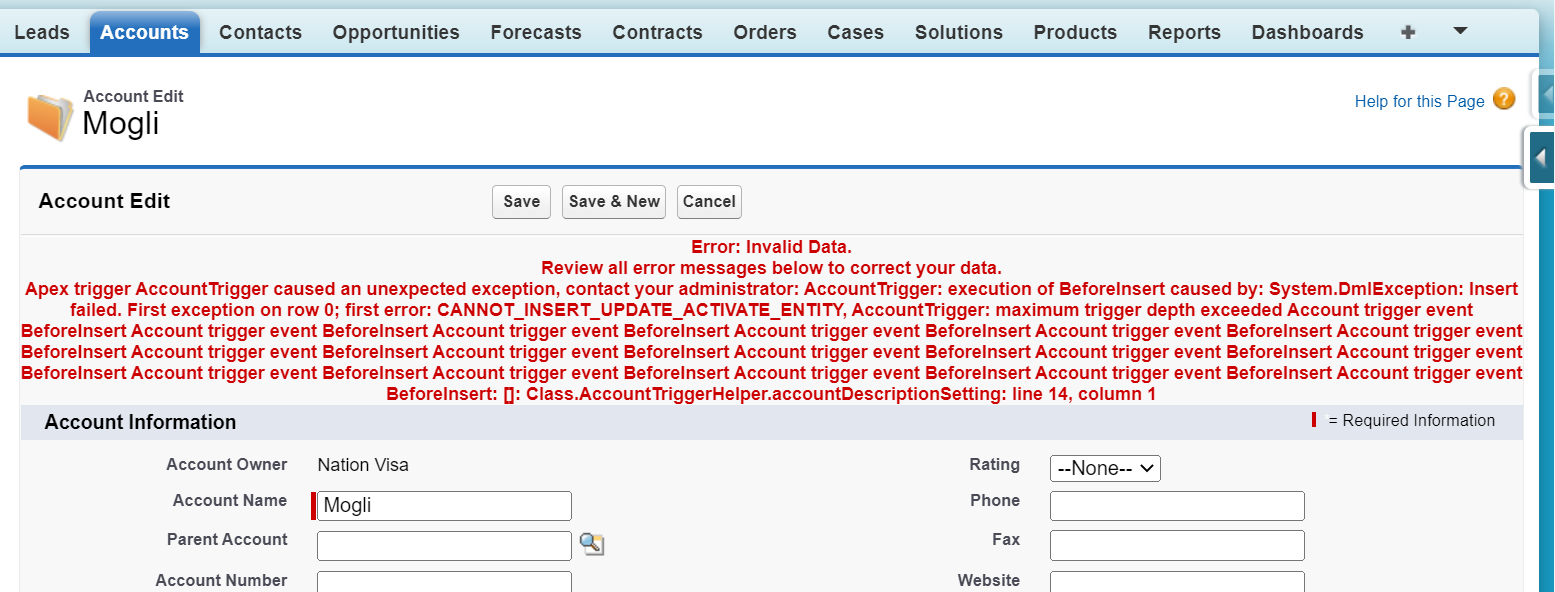
When a trigger calls to itself again and again (Generally **16 times depth**), then the concept is known as “Recursive Trigger”.



Salesforce Multi-Tenant Environment



When Trigger Recursion happens:



trigger AccountTrigger on Account (before insert) {

if(trigger.isInsert && trigger.isBefore){

AccountTriggerHandler.beforeInsert(trigger.new);

}

}

public class AccountTriggerHandler {

public static void beforeInsert(List<Account> accNewList){

AccountTriggerHelper.accountDescriptionSetting(accNewList);

}

}

public class AccountTriggerHelper {

public static void accountDescriptionSetting(List<Account> accNewList){

List<Account> accList = new List<Account>();

for(Account objAcc : accNewList){//Pop

Account newAcc = new Account(Name='Cinemax404');

accList.add(newAcc);

}

if(!accList.isEmpty()){

insert accList;

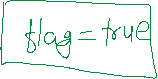
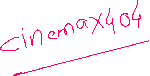
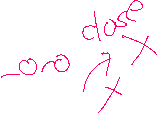
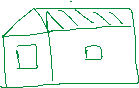
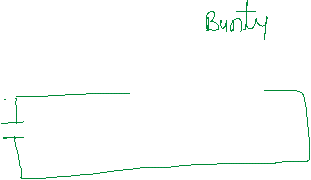
System.debug('Inside of insert');

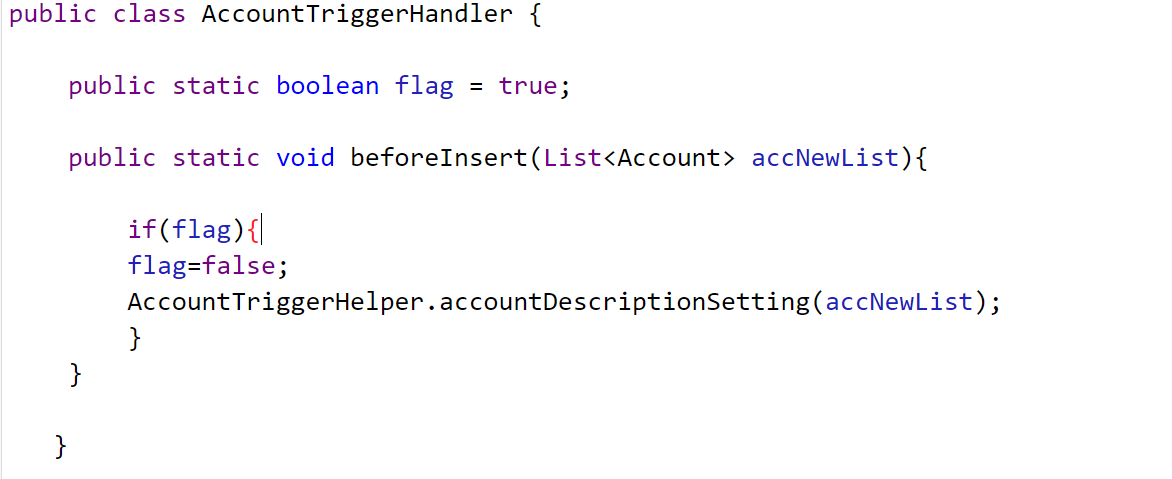
}

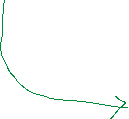
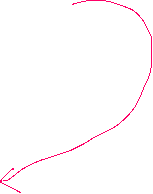
}

}

How to Prevent Recursive Trigger:







trigger AccountTrigger on Account (before insert) {

if(trigger.isInsert && trigger.isBefore){

AccountTriggerHandler.beforeInsert(trigger.new);

}

}

public class AccountTriggerHandler {

public static boolean flag = true;

public static void beforeInsert(List<Account> accNewList){

if(flag){

flag=false;

AccountTriggerHelper.accountDescriptionSetting(accNewList);

}

}

}

public class AccountTriggerHelper {

public static void accountDescriptionSetting(List<Account> accNewList){

List<Account> accList = new List<Account>();

for(Account objAcc : accNewList){//Pop

Account newAcc = new Account(Name='Cinemax404');

accList.add(newAcc);

}

if(!accList.isEmpty()){

insert accList;

System.debug('Inside of insert');

}

}

}

**2] Record Sharing**

Record Sharing Types:

1] Manual Sharing

2] Sharing Rules (Automatic Sharing)

- Based on the Record Owner

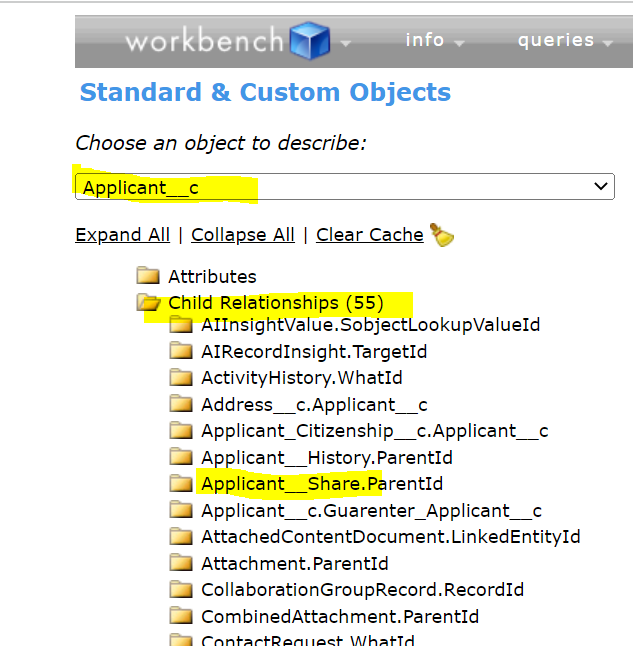
- Based on the Criteria

3] Roles and Hierarchies

4] Apex Sharing (Coding)

OWD = PRIVATE

|  |  |
| --- | --- |
| **Object** | **Sharing Object** |
| Account | AccountShare |
| Contact | ContactShare |
| Applicant\_\_c | Applicant\_\_Share |
| Passport\_\_c | Passport\_\_Share |



Account:

AccountShare accShare = new AccountShare(); // STEP 1 Instance

accShare.AccountId = '0015j00000dHaFt'; //STEP 2 Record Id

accShare.UserOrGroupId = '0055j0000049SY9'; //STEP 3 User ID

accShare.AccountAccessLevel = 'Read'; //or Edit

accShare.CaseAccessLevel = 'Read';

accShare.OpportunityAccessLevel = 'Read';

Database.insert(accShare,false);

Applicant\_\_c:

Applicant\_\_Share appShare = new Applicant\_\_Share();

appShare.ParentId = 'a005j00000B0mwS'; // Record Id

appShare.UserOrGroupId = '0055j0000049SY9'; // User ID

appShare.AccessLevel = 'Edit';

appShare.RowCause = Schema.Applicant\_\_Share.RowCause.Manual;

Database.insert(appShare,false);

User Story 1] Create a new Account. After creation, share this record with Bubli.

Event : After

Operation : Insert

Trigger : Account

Affecting Object : AccountShare

trigger AccountSharingTrigger on Account (after insert) {

List<AccountShare> accShareList = new List<AccountShare>();

for(Account objAcc : trigger.new){

AccountShare accShare = new AccountShare(); // STEP 1 Instance

accShare.AccountId = objAcc.Id; //STEP 2 Record Id to be shared

accShare.UserOrGroupId = '0055j000005SQyY'; //STEP 3 User ID

accShare.AccountAccessLevel = 'Read'; //or Edit

accShare.CaseAccessLevel = 'Read';

accShare.OpportunityAccessLevel = 'Read';

accShareList.add(accShare);

}

if(!accShareList.isEmpty())

Database.insert(accShareList,false);

}

User Story 2] Create a Lookup relationship with User Object for Account. Share the Account record with the selected user.

trigger AccountSharingTrigger on Account (after insert) {

List<AccountShare> accShareList = new List<AccountShare>();

for(Account objAcc : trigger.new){

AccountShare accShare = new AccountShare(); // STEP 1 Instance

accShare.AccountId = objAcc.Id; //STEP 2 Record Id to be shared

accShare.UserOrGroupId =objAcc.User\_\_c ; //STEP 3 User ID

accShare.AccountAccessLevel = 'Read'; //or Edit

accShare.CaseAccessLevel = 'Read';

accShare.OpportunityAccessLevel = 'Read';

accShareList.add(accShare);

}

if(!accShareList.isEmpty())

Database.insert(accShareList,false);

}

==============================END======================

Interview:

1] What is a trigger

2] Best practices of the trigger.

3] What is a recursive trigger and how to prevent it.

4] Where have you used the trigger in your project.

5] Coding Test

1) One Trigger Per Object

Always prefer to have one trigger per object. In case of multiple triggers per object, you don’t have control over the order of execution for triggers.

2) Logic-less Triggers

Avoid writing logic/method in the trigger. Logic/method written inside trigger cannot be exposed to other class. Thus, it decreased code reusability and not able to write test class properly.

3) Context-Specific Handler Methods

Create context-specific handler methods in Trigger handlers like afterInsertLogic, beforeInsertLogic etc.

4) Bulkify your Code

Always prefer to Bulkify your trigger code. It should work as designated when there are more than one record.

5) Avoid using DML statements and SOQL Queries inside FOR Loops

A single Apex transaction gets a maximum of 100 SOQL queries before exceeding the governor limit. So, if the trigger is invoked by a batch of more than 100 records, the governor limit will throw a runtime exception. Also, try to query related objects in one single query.

6) Using Collections, Streamlining Queries, and Efficient For Loops

It is important to use Apex Collections to efficiently query data and store the data in memory. A combination of using collections and streamlining SOQL queries can substantially help writing efficient Apex code and avoid governor limits.

7) Querying Large Data Sets

The total number of records that can be returned by SOQL queries in a request is 50,000. If returning a large set of queries causes you to exceed your heap limit, then a SOQL query for loop must be used instead. It can process multiple batches of records through the use of internal calls to query and query more.

8) Use @future Appropriately

It is essential to write down your Apex code with efficiency handle bulk or many records at a time. Use @fututre just in case of a call out from trigger to make sure the current transaction should not have to wait for a call out response.

9) Avoid Hardcoding IDs

When deploying Apex code between different environments or Sandbox, or installing Force.com AppExchange packages, it is essential to avoid hardcoding any kind of value such as IDs in the Apex code. If possible, use custom label/custom setting to store such hardcoded values.